AI Agents 101 - 2-page cheat sheet

Free course companion - the loop, tools, memory, safety, and the build checklists.

What makes an agent an agent

- **The distinction:** A chatbot answers your messages. An agent pursues a goal - it decides what to do next (search, read, compare, search again) until the goal is met or it gets stuck.

- **1. Autonomy:** Takes multiple steps without you steering each one.

- **2. Tools:** Can act on the world - search, read files, run code, call APIs - not just talk.

- **3. Goals + success criteria:** Knows what 'done' looks like, so it can tell progress from wandering.

- **All three or nothing:** Remove any one and you're back to a chatbot with extra steps. Mental shift: with a chatbot YOU are the loop; with an agent, the agent is.

The agent loop

- **Perceive:** Look at the goal and what's happened so far.

- **Reason:** Decide the next step.

- **Act:** Use a tool or produce output.

Perceive -> reason -> act -> repeat, until the goal is met or a limit is hit. If you understand this loop, you understand agents. Everything else is elaboration.

How tool calling works (the mechanism)

- **1. You describe the tools:** Each tool gets a name, plain-English description, and argument schema. These go into the model's context with your goal.

- **2. The model ASKS:** It outputs a structured request like 'call search_web with query=...'. It executes nothing - it's still generating text, in a format your code recognizes.

- **3. Your code RUNS the tool:** Your program sees the request, calls the real API, gets real results.

- **4. Results go back in:** Your code appends 'search_web returned: [...]' to the conversation; the model continues.

- **Key:** The model never calls code directly. Debugging agents means debugging this handoff.

Designing good tools

- **Narrow:** One thing per tool. search_web(query) is good; do_research(topic, depth, sources, format) is a trap - the model guesses what 'depth' means and your code interprets four fuzzy args. If it needs a paragraph to explain, split it.

- **Named like verbs:** The model picks tools by reading names. get_weather beats weather.

- **Documented like contracts:** What it does, what it needs, what it returns. Smart-intern test: would a smart intern know when and how to use it from the description alone?

Where agents shine - and flop

- **Shine:** Open-ended research with clear success criteria; multi-step digital chores; coding tasks with tests (compiler as perceiver); anything where try-check-adjust beats get-it-right-first-time. They excel when wrong steps are cheap and checking is cheap.

- **Flop:** Irreversible actions (money, delete, publish); expertise/taste (confident mediocrity); long-horizon planning (errors compound silently). Plus economics: 50 steps costs ~50x one answer - budget steps like money.

Safety basics

- **1. Least privilege:** A research agent needs web search, not your email password. A coding agent needs one project folder, not your whole drive. Every permission not granted is a failure mode you don't have.

- **2. Human approval:** Sending, paying, deleting, publishing - anything irreversible pauses the loop and asks a human. Build as a CODE gate, not a prompt suggestion: the model doesn't get a vote.

- **3. Sandbox:** Test folder, trial account, staging API. An agent that can only break a sandbox can only teach you things.

- **4. Log everything:** Every call, every argument, every result. When something goes wrong, the log answers 'what did it actually do?' Never run an agent you can't audit.

- **Meta-rule:** Start maximally restricted; loosen one permission at a time, each for a stated reason.

Short-term memory = the transcript

- **How it remembers:** Every loop iteration re-reads the whole conversation: goal, past reasoning, every tool call and result. Step 4 is informed by steps 1-3 because they're literally still on the page.

- **Context window:** Max text the model can consider at once (a token ~= 3/4 word). When the window fills, oldest content is cut - the agent 'forgets' the goal, repeats searches, contradicts findings. Looping in circles? Suspect the window first.

- **Keep outputs small:** Return summaries, not raw 50-page dumps - every tool result taxes the window on every future step.

- **Watch step count:** The step limit isn't just about loops - it's a memory budget. Fewer steps, less transcript, more room to think.

Long-term memory: notes first

- **Notes file:** After each run, append what was learned ('prefers morning briefings under 200 words'). Next run reads the file first. Crude, transparent, debuggable - sufficient for most personal agents. Start here; fancy comes later.

- **Scale up:** At hundreds of entries, switch to a vector store: retrieves by meaning, not keywords.

RAG: retrieval-augmented generation

- **1. Search:** Turn the question into a query; find candidates by keyword, meaning, or both.

- **2. Select:** Pick the few most relevant chunks - not whole documents. Also your context-window defense.

- **3. Ground:** 'Answer using ONLY these sources. Cite which chunk each claim comes from. If the answer isn't there, say so.'

Build 1: research agent (the plan)

- **Goal (one sentence):** 'Given a research question, produce a 5-bullet briefing with at least 3 cited sources, in under 10 tool calls.' Numbers make success objective.

- **Tool 1:** web_search(query) -> {title, url, snippet}. Read-only, auto-run.

- **Tool 2:** read_page(url, question) -> page summarized against the question, max ~300 words. Your context defense, built into the tool.

- **The loop:** ~80 lines of Python. Any chat model with an OpenAI-compatible API. Key and endpoint marked in code - never commit the key.

```
SYSTEM: research agent. Goal: 5-bullet briefing,
>=3 cited sources. Tools: web_search(query),
read_page(url, question). Think step by step; call
one tool as {"tool": name, "args": {...}} or finish
with {"tool": "finish", ...}. Max 10 steps.
Never invent sources.
```

Test adversarially (Build 1 hardening)

- **1. Junk test:** Point it at a topic with terrible search results. Does it finish honestly with 'I could only verify X' - or polish garbage into a confident briefing? Strengthen grounding: 'cite every claim; say unverified otherwise.'

- **2. Loop test:** Ask for the 2027 revenue of a company that doesn't exist. It should burn a few steps, then stop - via the step limit, or by recognizing futility and finishing with a partial answer.

- **3. Context test:** Huge question, max_steps=10. Does it still remember the question at step 9? If not, tighten the summary cap or add mid-run compaction.

- **4. Citation audit:** Open every cited source of a real briefing. Does each bullet follow from its citation? Five minutes; catches the confident-wrongness no automated check finds.

Every failure becomes a rule in the system prompt or a check in the code.

Build 2: your own task agent (design doc)

- **1. The job:** Goal sentence with numbers, again.

- **2. The tools:** New vs reused from Build 1. Most personal agents need 2-4 tools; if your list has eight, the task is too big. Cut scope, not corners.

- **3. Human gates:** Which auto-run, which need approval, what's forbidden. Write the actual UX: 'the agent pauses and shows me X, I reply yes/no.'

Next steps (honest map)

- **Multi-agent:** Specialists beat generalists - but failure modes multiply. Rule: don't go multi-agent until a single agent provably can't do the job. Start with two agents and a strict handoff format, not a swarm.

- **Production checklist:** Evals (fixed test tasks run after every change), approval gates, cost caps, logging.